

可靠性信息管理和分析

**系统概要设计说明书**

**广州易宝信息技术有限公司**

**版权所有不得复制**

**修订历史记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本号** | **发布日期** | **编写人** | **审阅人** | **更新说明** |
| V0.1 | 2018-4-24 | 黄鹏杰 |  | 创建 |
| V1.0 | 2018-5-17 | 伍志坚 | 凌征泉 | 修改 |
|  |  |  |  |  |
|  |  |  |  |  |

**目录**

[1 引言 5](#_Toc514828527)

[1.1 标识 5](#_Toc514828528)

[1.2 编写目的 5](#_Toc514828529)

[1.3 背景 5](#_Toc514828530)

[1.4 定义、缩写词和缩略语 6](#_Toc514828531)

[1.5 参考资料 6](#_Toc514828532)

[2 总体设计 6](#_Toc514828533)

[2.1 设计原则 6](#_Toc514828534)

[2.2 设计约束 7](#_Toc514828535)

[2.2.1 关键需求 7](#_Toc514828536)

[2.2.2 运行环境 8](#_Toc514828537)

[2.2.3 设计目标 8](#_Toc514828538)

[2.3 系统结构 8](#_Toc514828539)

[2.3.1 业务架构 8](#_Toc514828540)

[2.3.2 业务技术架构 10](#_Toc514828541)

[2.4 系统设计 10](#_Toc514828542)

[2.4.1 系统架构 10](#_Toc514828543)

[2.4.2 项目功能模块（微服务模块） 11](#_Toc514828544)

[2.4.3 第三方服务接口 12](#_Toc514828545)

[2.4.4 服务器配置清单 13](#_Toc514828546)

[2.4.5 数据流 15](#_Toc514828547)

[2.4.6 第三方服务接口清单 17](#_Toc514828548)

[状态监测接口 17](#_Toc514828549)

[故障诊断接口 17](#_Toc514828550)

[寿命预测接口 18](#_Toc514828551)

[健康评估接口 19](#_Toc514828552)

[维修决策接口 20](#_Toc514828553)

[根据故障代码查询故障详述接口 20](#_Toc514828554)

[根据产品结构查询FMEA故障关联层级信息的接口 21](#_Toc514828555)

[设备信息查询接口 21](#_Toc514828556)

[2.4.7 服务注册 22](#_Toc514828557)

[2.4.8 文件共享 22](#_Toc514828558)

[2.4.9 数据库大量数据获取 22](#_Toc514828559)

[2.5 开发环境 22](#_Toc514828560)

[2.5.1 硬件环境 22](#_Toc514828561)

[2.5.2 软件环境 23](#_Toc514828562)

[2.5.3 开发工具 23](#_Toc514828563)

[2.6 文件管理 23](#_Toc514828564)

[2.6.1 源代码文件 23](#_Toc514828565)

[2.7 公用模块设计 24](#_Toc514828566)

[2.8 Docker部署策略 25](#_Toc514828567)

[2.8.1 微服务容器化 25](#_Toc514828568)

[2.8.2 部署策略 27](#_Toc514828569)

[3 软件架构设计 28](#_Toc514828570)

[3.1 访问数据库决策 28](#_Toc514828571)

[3.2 软件架构的具体实现 28](#_Toc514828572)

[3.2.1 控制层 29](#_Toc514828573)

[3.2.2 业务层 29](#_Toc514828574)

[3.2.3 数据持久层 29](#_Toc514828575)

[3.2.4 视图层 30](#_Toc514828576)

[3.3 数据库设计原则 30](#_Toc514828577)

[3.3.1 索引设计 30](#_Toc514828578)

[3.3.2 表空间设计 31](#_Toc514828579)

[3.3.3 数据完整性设计 31](#_Toc514828580)

[4 体系结构设计 33](#_Toc514828581)

[4.1 需求规定 33](#_Toc514828582)

[4.2 结构 33](#_Toc514828583)

[4.3 关键技术设计 33](#_Toc514828584)

[4.3.1 安全性设计原则 33](#_Toc514828585)

[4.3.2 安全性功能设计 34](#_Toc514828586)

[5 界面设计原则 34](#_Toc514828587)

[5.1 简易性 34](#_Toc514828588)

[5.2 用户的语言 34](#_Toc514828589)

[5.3 记忆负担最小化 34](#_Toc514828590)

[5.4 一致性 35](#_Toc514828591)

[6 运行设计 35](#_Toc514828592)

[6.1 系统初始化 35](#_Toc514828593)

[6.2 运行控制 35](#_Toc514828594)

[6.3 运行结束 35](#_Toc514828595)

[7 系统性能 35](#_Toc514828596)

[7.1 并发性的考虑 35](#_Toc514828597)

[8 系统出错处理设计 36](#_Toc514828598)

[8.1 出错信息 36](#_Toc514828599)

[8.2 补救措施 36](#_Toc514828600)

[8.2.1 数据库备份 36](#_Toc514828601)

[8.3 系统维护设计 36](#_Toc514828602)

[8.3.1 环境变量维护 36](#_Toc514828603)

[9 编码规范 37](#_Toc514828604)

[9.1 命名规范 37](#_Toc514828605)

[9.1.1 日志记录 37](#_Toc514828606)

[9.1.2 Java包名的命名规范 37](#_Toc514828607)

[9.1.3 JSP/HTML命名规范 37](#_Toc514828608)

[9.2 Java程序代码开规则 39](#_Toc514828609)

[9.2.1 通用规则 39](#_Toc514828610)

[9.2.2 Import规则 40](#_Toc514828611)

[9.3 Java编程风格 41](#_Toc514828612)

[9.3.1 缩进与空格 41](#_Toc514828613)

[9.3.2 大括号和嵌套风格 42](#_Toc514828614)

[9.3.3 类的声明 46](#_Toc514828615)

[9.3.4 例外处理 47](#_Toc514828616)

[9.3.5 语句的处理 47](#_Toc514828617)

[9.4 注释 48](#_Toc514828618)

[9.4.1 注释规则 48](#_Toc514828619)

[10 微服务 49](#_Toc514828620)

[10.1 什么是微服务 49](#_Toc514828621)

[10.2 微服务设计原则 49](#_Toc514828622)

[10.3 微服务优势与缺点 50](#_Toc514828623)

[11.3.1 微服务优势 50](#_Toc514828624)

[11.3.2 微服务缺点 50](#_Toc514828625)

# 引言

## 标识

|  |  |
| --- | --- |
| 中文全称 | 可靠性信息管理和分析系统 |
| 中文简称 | 可靠性信息系统 |
| 英文全称 |  |
| 英文简称 |  |

## 编写目的

本文档是《可靠性信息管理和分析系统》升级概要设计说明书，在用户需求分析明确了系统的业务及流程的基础上，对系统功能进行了概要性的架构及接口设计、业务逻辑及展现层设计，并对系统性能及编码规范要求进行了描述。系统概要设计是系统设计的重要组成部分，也是详细设计的基础。本《系统概要设计》预期读者包括：

* 建设方技术及管理人员
* 承建方系统设计、开发人员

## 背景

质量与可靠性信息管理是深入开展质量、可靠性、维修性、保障性管理工作的重要环节，贯穿于产品的设计、生产、试验和使用维护等全寿命周期的各个阶段，对监测、评价过程控制和质量与可靠性水平的高低起着重要的作用，是改进产品质量与可靠性水平的高低起着重要的作用，是改进产品质量与可靠性、改善产品研制各环节工作质量的最直接的原始资料和依据，也是正确认识影响产品质量特性诸因素变化、产品质量波动的内在联系和掌握提高产品质量规律性的基本依据。此外，对管理者进行质量决策提供参考。

综上所述，本系统为可靠性信息管理和分析系统，旨在对工业设备可靠性信息进行管理和分析，达到控制产品质量的目的。

## 定义、缩写词和缩略语

* "可靠性信息管理和分析系统"以下简称"可靠性信息系统"。
* “工业和信息化部电子第五研究所”以下简称”五所”。

## 参考资料

* + 《计算机软件文档编制规范 GB/T 8567-2006》
  + 《GB/T 8566－2007》
  + 《可靠性信息管理和分析系统项目\_需求规格说明书V1.0.doc》

# 总体设计

## 设计原则

1. 先进性。应采用国际先进的技术路线和体系结构。具有先进的技术水平，采用标准化产品，有较高的性能，符合当今世界技术发展的方向。
2. 可靠性。系统应能保证每周七天、每天二十四小时持续可靠运行，用户在使用应用系统的过程中，系统平均出错率应保持在0.1%（由于用户误操作引起的除外）。
3. 易操作性。系统设计过程必须考虑到易于操作、易于使用、界面友好等问题，不培训或只需少量的培训即可使用。
4. 开放性。系统的设计和建设必须具有开发性，应充分考虑网络、硬件的扩展，必须能跨平台运行，如UNIX、 Linux、Windows NT等平台。
5. 实用性。系统应坚持以需求为导向，系统应可方便、简单地供业务人员使用。系统应有相应的措施，以帮助用户避免操作上的错误。系统上所有的词汇、语言要求简单明了、无歧义。所有的用户在短时间的培训后就能使用系统。
6. 可扩展性。系统必须提供标准的开发接口与将来扩展的B/S和C/S结构的业务系统集成，特别要加强系统设计的前瞻性、预留系统扩充和扩展能力，在不影响其它系统运行的情况下，进行系统的平滑升级。
7. 安全性。系统数据必须安全，每一个操作都可以有日志可查，因此，系统必须能够提供有效的安全保密措施，确保整个系统的安全运行。
8. 集成性。系统可与其他业务系统方便集成。
9. 规范性。系统必须严格符合主流的技术标准，特别是SOA技术规范。
10. 易维护性。系统必须支持二次开发，提供良好的接口和完善的技术文档；高度模块化，方便项目交接。
11. 技术支持性：系统要求提供多方便的技术支持，包括在线帮助、用户手册、管理员手册等。

## 设计约束

### 关键需求

* + 采用J2EE三层结构；
  + 采用springcloud实现微服务
  + 采用zuul实现api网关
  + 采用ribbon负载均衡实现服务高可用
  + 采用mongodb与mysql实现数据存储
  + 使用Hystrix实现微服务的容错处理
  + 采用rabbitmq实现服务之间通信
  + 本系统需要与工业设备端进行数据交互
  + 采用webSocket进行数据实时监测数据推送
  + 采用docker部署和发布微服务

### 运行环境

本系统的使用者为企业用户、维护人员、五所领导、五所工程师、设备管理员，因此在服务器配置上要求比较高、网络环境还必须稳定。

服务器要求：

|  |  |  |
| --- | --- | --- |
| **服务器** | **系统软件类型** | **采用的系统软件** |
| 数据服务器 | 操作系统 | Linux |
| 数据库系统 | Mysql、mongodb |
| 应用服务器 | 操作系统 | Linux |
| APP Server | Tomcat 8.0 |

### 设计目标

在设计原则的指导思想下，实现以下目标。

* + 实现展现层、支持层、数据层的分层设计，以利于项目开发的系统分析、编码实现、性能监控调整、灵活扩展等；
  + 平衡各个系统层次的压力；
  + 重点把握关键的业务需求和实现的技术难点；
  + 统一系统设计、系统开发过程、内外部接口标准的规范和管理；
  + 保障展现层的可操作性，满足用户的实际需求；

## 系统结构

### 业务架构

本系统总体设计如下图所示。

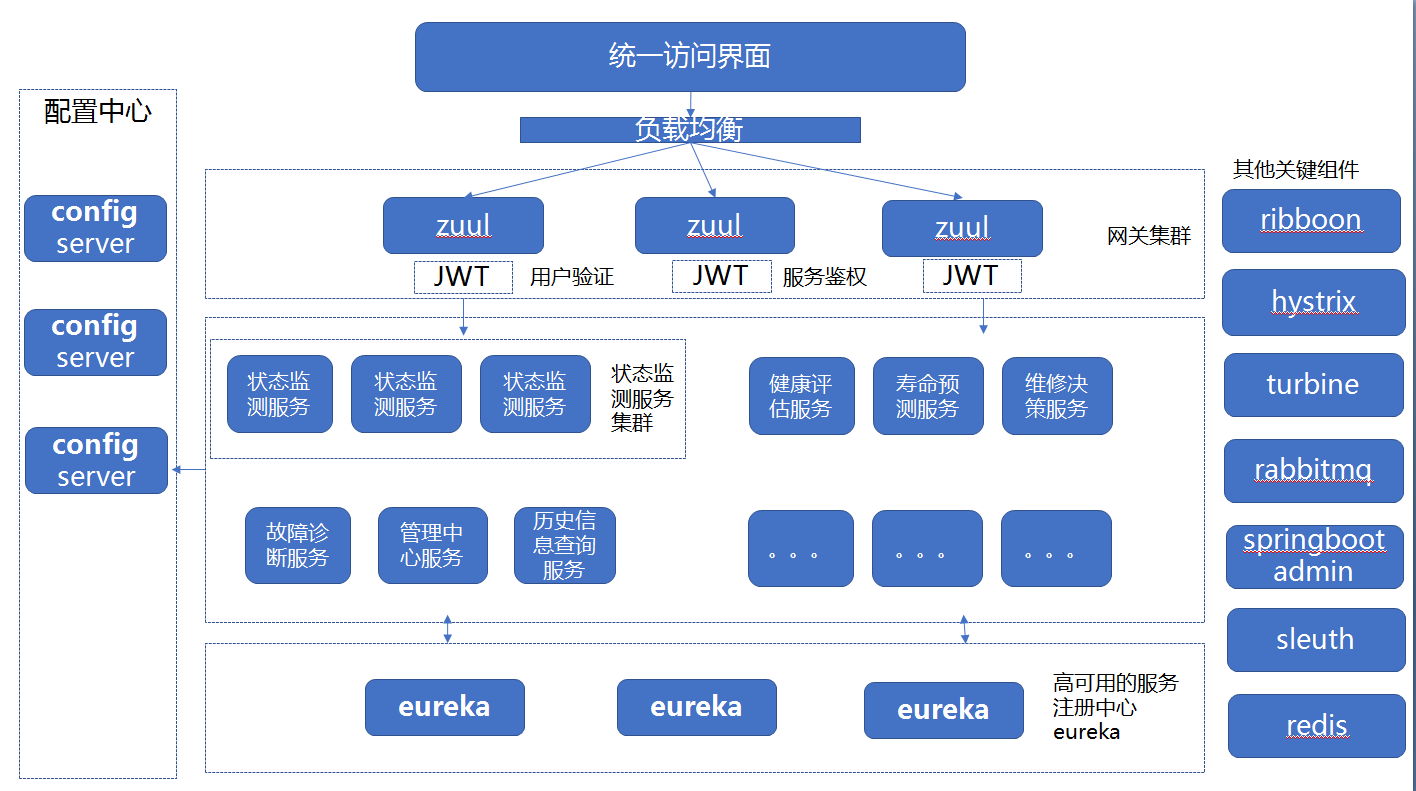


### 业务技术架构

* **采集层**：采用统一的采集平台接收传感器上传的数据，每个行业每类数据采用mongodb保存数据。
* **业务层**：将提供服务的配置库按模块进行微服务划分，后续数据量大可以进一步的微服务化。
* **展现层**：采用统一的页面管理。

## 系统设计

### 系统架构



前端浏览器的请求首先到达服务网关（zuul集群），在服务网关对用户权限进行合法性验证、系统日志，然后再到具体的服务，服务统一注册到高可用的服务注册中心集群，服务之间可以相互调用，服务的所有的配置文件由配置服务管理，配置服务的配置文件放在git仓库，方便开发人员随时改配置。

### 项目功能模块（微服务模块）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 服务模块模块名称 | 功能描述 | 备注 |
| 1 | monitor-service 状态监测服务 | 第三方的状态监测服务，在本系统中支持集群部署多个服务，由feign-consume实现负载均衡。 |  |
| 2 | predict-service 寿命预测服务 | 第三方的寿命预测服务，在本系统中支持集群部署多个服务，由feign-consume实现负载均衡。 |  |
| 3 | evaluate-service 健康评估服务 | 第三方的健康评估服务，在本系统中可以集群部署多个服务，由feign-consume实现负载均衡。 |  |
| 4 | diagnose-service 故障诊断服务 | 第三方的故障诊断服务，在本系统中可以集群部署多个服务，由feign-consume实现负载均衡。 |  |
| 5 | decision-service 维修决策服务 | 第三方的维修决策服务，在本系统中可以集群部署多个服务，由feign-consume实现负载均衡。 |  |
| 6 | manage-service 管理中心服务 | 包括用户注册、用户管理、角色管理、权限管理、基础数据等后台管理功能模块、案例数据的生成。 |  |
| 7 | History-service  历史信息查询 | 状态监测、寿命预测、健康评估、故障诊断、维修决策等历史数据的查询统计 |  |
| 8 | zuul-server 代理负载网关 | api网关集群，实现反向代理和客户端负载均衡。网关负责统一接收所有请求，然后根据请求的类型进行转发到不同的服务。使用网关能够统一的管理客户端的请求日志、进行权限控制、过滤等 |  |
| 9 | feign-consume 负载均衡 | 负责调用第三方服务、响应前端，负责第三方服务的具体api管理 |  |
| 10 | eureka-server 注册中心 | 服务的注册中心，第三方的状态监测服务、寿命预测服务、健康评估服务、故障诊断服务、维修决策服务在这里注册。管理服务的地址、端口、服务类型（如http://127.0.0.1:8090/） |  |
| 11 | admin-server 应用架构状态检测 | 服务的状态检测，利用Spring Boot Admin 来监控各个独立服务的运行状态，包括内存信息、JVM信息、垃圾回收信息等 |  |
| 12 | sleuth-server 服务的状态监控 | 第三方服务之间的监控服务，可以通过该模块图形化了解第三方服务之间调研的情况（日志查看） |  |
| 13 | hystrix-dashboard | 图形化了解单个应用内的服务信息，包括请求响应时间, 请求成功率等数据 |  |
| 14 | turbine-server 监控服务 | 监控服务集群，汇聚监控信息，并将汇聚到的监控信息提供给Hystrix Dashboard来集中展示和监控。 |  |
| 15 | config-server 配置中心 | 统一的管理分布式系统的配置中心, 支持在Git, SVN和本地存放配置文件, 方便开发人员随时改配置。 |  |
| 16 | 可靠性信息管理和分析系统（现有） | 五所的管理员和学生端的功能，这部分不采用微服务，使用现有系统，增加企业用户的审核、知识库管理、增加权限控制，控制算法的使用人、控制模型库的使用人、控制配置库的使用人 |  |
| 17 | 统一访问界面 | 实现五所用户、企业用户、单个设备的状态监测、寿命预测、健康评估、故障诊断、维修决策功能展示，以及历史信息的统计查询。 |  |

### 第三方服务接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 服务模块模块名称 | 功能接口 | 备注 |
| 1 | monitor-service 状态监测服务 | 1、状态监测服务接口 | 第三方服务提供 |
| 2 | predict-service 寿命预测服务 | 1、寿命预测服务接口 | 第三方服务提供 |
| 3 | evaluate-service 健康评估服务 | 1、健康评估服务接口 | 第三方服务提供 |
| 4 | diagnose-service 故障诊断服务 | 1、故障诊断服务接口 2、根据故障代码查询故障详述接口 | 第三方服务提供 |
| 5 | decision-service 维修决策服务 | 1、维修决策服务接口 2、根据产品结构查询FMEA故障关联层级信息的接口 | 第三方服务提供 |
| 6 | manage-service 企业管理中心服务 | 1、设备管理接口 2、人员管理接口 3、设备参数接口 | 自建 |
| 7 | History-service 历史信息查询 | 1、状态监测历史信息查询统计 2、寿命预测历史信息查询统计 3、健康评估历史信息查询统计 4、故障诊断历史信息查询统计 5、维修决策历史信息查询统计 | 自建 |

### 服务器配置清单

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 服务模块模块名称 | 服务器配置 | 备注 |
| 1 | monitor-service 状态监测服务 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：1T内存 |  |
| 2 | predict-service 寿命预测服务 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：1T内存 |  |
| 3 | evaluate-service 健康评估服务 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：1T内存 |  |
| 4 | diagnose-service 故障诊断服务 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：1T内存 |  |
| 5 | decision-service 维修决策服务 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：1T内存 |  |
| 6 | manage-service 管理中心服务 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |
| 7 | History-service 历史信息查询 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：1T内存 |  |
| 8 | zuul-server 代理负载网关 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |
| 9 | feign-consume 负载均衡 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |
| 10 | eureka-server 注册中心 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |
| 11 | admin-server 应用架构状态检测 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |
| 12 | sleuth-server 服务的状态监控 |  |
| 13 | hystrix-dashboard |  |
| 14 | turbine-server 监控服务 |  |
| 15 | config-server 配置中心 | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |
| 16 | 可靠性信息管理和分析系统（现有） | CPU：二颗2.00GHz八核处理器 内存：8G 硬盘：500G内存 |  |

### 数据流



### 第三方服务接口清单

#### 状态监测接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 | POST/PUT |
| 内容 | 将一段时间内的传感器上传的数据调用状态监测服务，状态监测算法将结果保存到文件中，服务返回文件路径，对文件进行解析后保存到数据库中，形成历史信息。并将解析后的数据在页面展示。 |
| 说明 |  |
| 请求参数 | 后期定义 |
| 返回 | 文件的内容，以Excel或者TXT文件格式储存  文件第一行：ret 返回结果、c错误代码,类型string(参考1.1)、d 错误描述,类型string  文件明细内容第二行为表头  shijian时间（YYYY-MM-DD 24H:dd:ss）  danwei 单位  zhibiao二级指标  value数值 |
| 返回说明 |  |

#### 故障诊断接口

|  |  |
| --- | --- |
| 路径 | 后期提供 |
| 方法 | POST/PUT |
| 内容 | 将一段时间内的传感器上传的数据，调用故障诊断服务，故障诊断算法将结果保存到文件中，服务返回文件路径，对返回的文件进行解析后保存到数据库中，形成历史信息。并将解析后的数据在页面展示。 |
| 说明 |  |
| 请求参数 | 后期定义 |
| 返回消息 |  |
| 返回说明 | 文件的内容，以Excel或者TXT文件格式储存  文件第一行：ret 返回结果、c错误代码,类型string(参考1.1)、d 错误描述,类型string  文件明细内容第二行为表头  gzgailv 故障概率类型string（0.12）  gzcode故障代码类型string |

#### 寿命预测接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 | POST/PUT |
| 内容 | 将一段时间内的传感器上传的数据，调用寿命预测服务，寿命预测算法将结果保存到文件中，服务返回文件路径，对返回的文件进行解析后保存到数据库中，形成历史信息。并将解析后的数据在页面展示。 |
| 说明 |  |
| 请求参数 | 后期定义 |
| 返回消息 |  |
| 返回说明 | 文件的内容，以Excel或者TXT文件格式储存  文件第一行：ret 返回结果、c错误代码,类型string(参考1.1)、d 错误描述,类型string  文件明细内容glmd表头  glmdX概率密度横坐标，类型string  glmdY概率密度纵坐标，类型string  文件明细内容tx表头  t类型string  x类型string  文件明细内容ty表头  t实时点坐标,类型string  y实时点坐标,类型string |

#### 健康评估接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 | POST/PUT |
| 内容 | 将一段时间内的传感器上传的数据，调用健康评估服务，健康评估算法将结果保存到文件中，服务返回文件路径，对返回的文件进行解析后保存到数据库中，形成历史信息。并将解析后的数据在页面展示。 |
| 说明 |  |
| 请求参数 | 后期定义 |
| 返回消息 |  |
| 返回说明 | 文件的内容，以Excel或者TXT文件格式储存  文件第一行：ret 返回结果、c错误代码,类型string(参考1.1)、d 错误描述,类型string  文件明细内容第二行为表头  p概率string（0.12）  l 健康等级 类型string |

#### 维修决策接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 | POST/PUT |
| 内容 | 将一段时间内的传感器上传的数据，调用维修决策服务，维修决策算法将结果保存到文件中，服务返回文件路径，对返回的文件进行解析后保存到数据库中，形成历史信息。并将解析后的数据在页面展示。 |
| 说明 |  |
| 请求参数 | 后期定义 |
| 返回消息 |  |
| 返回说明 | 文件的内容，以Excel或者TXT文件格式储存  文件第一行：ret 返回结果、c错误代码,类型string(参考1.1)、d 错误描述,类型string  文件明细内容第二行为表头  p概率，类型string  s 故障代码，类型string  g 维修决策，类型string |

#### 根据故障代码查询故障详述接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 |  |
| 内容 | 根据故障代码查询故障详述接口 |
| 说明 |  |
| 请求参数 | gzcode |
| 返回消息 | 返回故障信息 |
| 返回说明 |  |

#### 根据产品结构查询FMEA故障关联层级信息的接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 |  |
| 内容 | 根据产品结构查询FMEA故障关联层级信息的接口 |
| 说明 |  |
| 请求参数 | gzcode |
| 返回消息 | 返回故障信息 |
| 返回说明 |  |

#### 设备信息查询接口

|  |  |
| --- | --- |
| 路径 | 部署后确定 |
| 方法 |  |
| 内容 | 查询设备台账、产品结构、功能性能的信息 |
| 说明 |  |
| 请求参数 | gzcode |
| 返回消息 | 返回设备台账、产品结构、功能性能的信息 |
| 返回说明 |  |

### 服务注册

第三方服务注册信息包括：

#服务端口号 server.port=

#服务名称 spring.application.name=

### 文件共享

服务之间有文件需要共享约定为： 则由发起请求时将文件路径以参数的形式传给相应的服务，相应的服务通过路径访问文件。

### 数据库大量数据获取

请求服务之前需要从数据库获取大量数据给matlab算法,则由发起请求时将数据查询出来并保存到文件，再将文件路径以参数的形式传给相应的服务，相应的服务通过路径访问文件。

## 开发环境

### 硬件环境

工作站：笔记本，CPU双核1.8G；内存2G以上；硬盘80G以上

2台数据库服务器

配置：二颗E5-2640八核处理器，2.00GHz ；64G内存；4个热插拔500GB SAS硬盘；

6台应用服务器

配置：二颗E5-2640八核处理器，2.00GHz ；32G内存；4个热插拔500GB SAS硬盘；

### 软件环境

* + 服务器操作系统
* Red Hat Enterprise Linux AS release 4
  + 数据库：
* Mysql、mongodb
  + 开发PC：
* Windows 7/10
* IE 9/10/11

### 开发工具

* + 版本控制：SVN
* 进度管理：Microsoft Project 2010
  + 设计工具：Microsoft Visio 2010、PowerDesiger、StarUML
  + 部署运行：Tomcat 8
  + 文档工具：Microsoft Word/Excel/PowerPoint 2010/WPS
  + 开发环境：Eclipse 、MyEclipse、JDK7

## 文件管理

### 源代码文件

系统的代码分为JAVA代码，页面代码、JS代码、样式代码和图片等。源代码的基础包名为src/main/java/cn/com/epro/模块名，在基础包的基础上，。

包命名规范：

|  |  |
| --- | --- |
| 包名 | 描述 |
| src/main/java/cn/com/epro/模块名 | 系统基础包名 |
| src/main/java/cn/com/epro /模块名/controller | 控制类包 |
| src/main/java/cn/com/epro/模块名/bean | 实体类包 |
| src/main/java/cn/com/epro/模块名/mapper | mysql数据访问类包 |
| src/main/java/cn/com/epro/模块名/repository | mongodb数据访问类包 |
| src/main/java/cn/com/epro/模块名/service | 业务逻辑类包 |
| src/main/java/cn/com/epro/模块名/util | 工具类包 |

页面文件及配置文件等的基础包名为src/main/resources,目录命名规范

|  |  |
| --- | --- |
| 名称 | 描述 |
| src/main/resources /public | 存放页面文件。 |
| src/main/resources /mapper | Mybatis 映射文件路径。 |
| src/main/resources/public /css | 存放模块服务css样式 |
| src/main/resources/public /js | 存放模块服务js脚本 |

## 公用模块设计

系统公用模块包括：

|  |  |  |
| --- | --- | --- |
| 模块名称 | 端口 | 简介 |
| eureka-server | 可灵活配置 | 服务注册中心，提供服务注册、发现功能 |
| sleuth-server | 为服务之间调用提供链路追踪。即了解一个服务请求经过了哪些服务，每个服务处理花费了多长等，方便理清各微服务间的调用关系 |
| zuul-server | Api网关，首先网关负责统一接收所有请求，然后根据不同的规则进行转发到不同的服务。再者，使用网关能够统一的管理请求日志、进行权限控制、过滤等，这样就能避免在每个单体应用中做重复的工作 |
| admin-server | 可以浏览所有被监控spring-boot项目的基本信息，详细的Health信息、内存信息、JVM信息、垃圾回收信息、各种配置信息（比如数据源、缓存列表和命中率）等 |
| common-module | 系统公用的工具类、配置、常量等 |
| hystrix-dashboard | 了解单个应用内的服务信息，包括请求响应时间, 请求成功率等数据 |
| turbine-server | 与hystrix-dashboard相比，特点是汇总系统内多个服务的数据 |
| conf-server |  | 统一的管理分布式系统的配置中心, 支持在Git, SVN和本地存放配置文件,为了能让客户端准确的找到配置文件。 |

## Docker部署策略

使用docker-compose将以下的微服务容器化，并进行自动化部署。

### 微服务容器化

**基础业务微服务：**

|  |  |  |
| --- | --- | --- |
| 模块名称 | 端口 | 简介 |
| monitor-server | 可灵活配置 | 服务提供者，提供状态监测服务 |
| predict-server | 服务提供者，提供寿命预测服务 |
| [evaluate](http://www.baidu.com/link?url=TrWBkAwPhMkw8gOb-hNxY2UX_fR6pjCFaAI_qSPnM91Wwtdpe49NglwWPmDD3hUDL8Jwue7xPXTKFYTk5uYdqjRXiasa-4EtPp9GblHXULS)-server | 服务提供者，提供健康评估服务 |
| diagnose-server | 服务提供者，提供故障诊断服务 |
| [decision](http://www.baidu.com/link?url=RaYGvjAJSJiQvMIbRPyglN7f2j33AWMSaKDqwSS3s0eS2wvM4r4K2LJEk6dPujWY_lH9Z5eOxKTXNDAiBXxmgHWS3z7KtD2npK-RoVteruuQRzmxmk-43vOMULKkckqv) -module | 服务提供者，提供维修决策服务 |
| front-app | 服务提供者的调用者，采用原始的RestTemplate调用，采用ribbon做客户端负载均衡，以及feign声明式服务调用 |

**基础公用微服务：**

|  |  |  |
| --- | --- | --- |
| 模块名称 | 端口 | 简介 |
| eureka-server | 可灵活配置 | 服务注册中心，提供服务注册、发现功能 |
| sleuth-server | 为服务之间调用提供链路追踪。即了解一个服务请求经过了哪些服务，每个服务处理花费了多长等，方便理清各微服务间的调用关系 |
| zuul-server | Api网关，首先网关负责统一接收所有请求，然后根据不同的规则进行转发到不同的服务。再者，使用网关能够统一的管理请求日志、进行权限控制、过滤等，这样就能避免在每个单体应用中做重复的工作 |
| admin-server | 可以浏览所有被监控spring-boot项目的基本信息，详细的Health信息、内存信息、JVM信息、垃圾回收信息、各种配置信息（比如数据源、缓存列表和命中率）等 |
| hystrix-dashboard | 了解单个应用内的服务信息，包括请求响应时间, 请求成功率等数据 |
| turbine-server | 与hystrix-dashboard相比，特点是汇总系统内多个服务的数据 |
| conf-server |  | 统一的管理分布式系统的配置中心, 支持在Git, SVN和本地存放配置文件,为了能让客户端准确的找到配置文件。 |

### 部署策略

**环境**：linux

**工具：docker+ docker registry**

原则：将基础业务微服务和基础公用微服务分开，主要是因为基础业务微服务依赖了基础公用微服务，部分基础业务微服务必须要等待基础公用微服务加载完毕才能正常启动，比如monitor-server项目使用了配置中心conf-server的配置，conf-server未加载完毕，monitor-server项目启动异常。

**步骤：**

1.先启动公用微服务,顺序如下：

eureka-server 服务注册/服务发现

admin-server

conf-server 配置更新/配置集中管理

zuul-server 智能路由/安全控制等

sleuth-server 服务跟踪

hystrix-dashboard

turbine-server 基于消息中间件的实时性能数据收集

common-module

2．再启动基础业务微服务，顺序如下：

monitor-server

predict-server

[evaluate](http://www.baidu.com/link?url=TrWBkAwPhMkw8gOb-hNxY2UX_fR6pjCFaAI_qSPnM91Wwtdpe49NglwWPmDD3hUDL8Jwue7xPXTKFYTk5uYdqjRXiasa-4EtPp9GblHXULS)-server

diagnose-server

[decision](http://www.baidu.com/link?url=RaYGvjAJSJiQvMIbRPyglN7f2j33AWMSaKDqwSS3s0eS2wvM4r4K2LJEk6dPujWY_lH9Z5eOxKTXNDAiBXxmgHWS3z7KtD2npK-RoVteruuQRzmxmk-43vOMULKkckqv) -module

front-app

# 软件架构设计

## 访问数据库决策

1.Springboot通过pom文件引入spring-boot-starter-data-mongodb依赖，写一个接口继承MongoRepository来查询，只需要定义一个方法即可，就可以实现对mongodb进行操作。

2.系统采用的mybatis是一个对JDBC进行轻量级封装的持久层框架，可以帮助简化JDBC的操作，包括获取数据库连接、设置参数、处理结果集、事务管理等等。通过XML配置或注解还可能可以非常方便的实现原本复杂的数据库操作。

## 软件架构的具体实现

基于springboot+mybatis+mongodb开发的应用架构分为四个层次，包括控制层、业务层，数据持久层、视图层和数据库，参见下图：



### 控制层

控制层的核心功能是处理跟用户交互的，一般每个业务逻辑都会有一个Controller，提供给用户http请求接口，用户请求接口进行数据访问。

### 业务层

Service层的核心就是编写业务逻辑并与mapper进行交互处理或获取数据。

### 数据持久层

数据持久层是通过mybatis对获取数据库连接、设置参数、处理结果集、事务管理

### 视图层

视图层的核心功能是用于处理页面和业务逻辑的交互，对数据进行必要的验证和转换，对数据进行呈现和对用户的非法操作进行适当的控制

## 数据库设计原则

1) 标准化和规范化

数据的标准化有助于消除数据库中的数据冗余。标准化有好几种形式，但Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，遵守3NF 标准的数据库的表设计原则是：“One Fact in One Place”即某个表只包括其本身基本的属性，当不是它们本身所具有的属性时需进行分解。表之间的关系通过外键相连接。它具有以下特点：有一组表专门存放通过键连接起来的关联数据。

事实上，为了效率的缘故，对表不进行标准化有时也是必要的。

2) 数据驱动

采用数据驱动而非硬编码的方式，许多策略变更和维护都会方便得多，大大增强系统的灵活性和扩展性。

3) 考虑各种变化

在设计数据库的时候考虑到哪些数据字段将来可能会发生变更。

举例，姓氏就是如此（注意是西方人的姓氏，比如女性结婚后从夫姓等）。所以，在建立系统存储客户信息时，在单独的一个数据表里存储姓氏字段，而且还附加起始日和终止日等字段，这样就可以跟踪这一数据条目的变化。

### 索引设计

索引是从数据库中获取数据的最高效方式之一。95%的数据库性能问题都可以采用索引技术得到解决。

1) 逻辑主键使用唯一的成组索引，对系统键（作为存储过程）采用唯一的非成组索引，对任何外键列采用非成组索引。考虑数据库的空间有多大，表如何进行访问，还有这些访问是否主要用作读写。

2) 大多数数据库都索引自动创建的主键字段，但是可别忘了索引外键，它们也是经常使用的键，比如运行查询显示主表和所有关联表的某条记录就用得上。

3) 不要索引memo/note 字段，不要索引大型字段（有很多字符），这样作会让索引占用太多的存储空间。

4) 不要索引常用的小型表

不要为小型数据表设置任何键，假如它们经常有插入和删除操作就更别这样作了。对这些插入和删除操作的索引维护可能比扫描表空间消耗更多的时间。

### 表空间设计

1. 表空间大小定义原则

当表空间 大小小于操作系统对最大文件限制时，表空间由一个文件组成。如果表空间大小大于操作系统对最大文件限制时，该表空间由多个数据文件组成，表空间的总大小为估算为:

　　Tablespace + sum (数据段+索引段)\*150%。

1. 表空间扩展性设计原则

表空间数据文件采用自动扩展的方式，扩展容量快大小按2的整数倍(1M、2M、4M、8M、16M、32M、64M)进行扩展，创建表空间时尽量采用nologing选项。表空间的最大限制一般采用unlimited，除非确切知道表空间数据文件的最大使用范围。(一般windows 32位系统的文件最大2G，64位的unix系统系统文件最大128G,但也要注意文件格式设定的文件大小),建议最大为2G。表空间采用local管理方式

### 数据完整性设计

1. 完整性实现机制：

实体完整性：主键

参照完整性：

父表中删除数据：级联删除；受限删除；置空值

父表中插入数据：受限插入；递归插入

父表中更新数据：级联更新；受限更新；置空值

DBMS对参照完整性可以有两种方法实现：外键实现机制（约束规则）和触发器实现机制

用户定义完整性：

NOT NULL；CHECK；触发器

1. 用约束而非商务规则强制数据完整性

采用数据库系统实现数据的完整性。这不但包括通过标准化实现的完整性而且还包括数据的功能性。在写数据的时候还可以增加触发器来保证数据的正确性。不要依赖于商务层保证数据完整性；它不能保证表之间（外键）的完整性所以不能强加于其他完整性规则之上。

1. 强制指示完整性

在有害数据进入数据库之前将其剔除。激活数据库系统的指示完整性特性。这样可以保持数据的清洁而能迫使开发人员投入更多的时间处理错误条件。

1. 使用查找控制数据完整性

控制数据完整性的最佳方式就是限制用户的选择。只要有可能都应该提供给用户一个清晰的价值列表供其选择。这样将减少键入代码的错误和误解同时提供数据的一致性。某些公共数据特别适合查找：国家代码、状态代码等。

1. 采用视图

为了在数据库和应用程序代码之间提供另一层抽象，可以为应用程序建立专门的视图而不必非要应用程序直接访问数据表。这样做还等于在处理数据库变更时给你提供了更多的自由。

# 体系结构设计

## 需求规定

详见《可靠性信息管理和分析系统项目\_需求规格说明书》

## 结构

## 关键技术设计

### 安全性设计原则

为了保证安全体系的实用性，在设计安全体系时，必须遵循以下几项原则：

* **安全可靠性**

用国内具有自主知识产权的密码算法和加密设备；

系统的运行和管理基于安全策略进行；

程序之间的数据传递通过安全数据通道进行。

* **标准化**

采用国密办批准的算法；

遵循相应的国际标准。

* **先进性**

及时采用国内最新研究成果，在满足当前业务的访问控制需求的同时，能够适应未来的技术发展趋势。

* **可扩展性**

采用分层结构，模块化设计；

模块之间和本系统与外部系统之间通过标准接口交互。

* **兼容性**

能够与主要的业务应用系统平台兼容。

* **经济性**

充分考虑节省资金。

### 安全性功能设计

* 操作员安全设计

操作员必须持有代表个人身份的用户名称和密码，经过身份认证后才能够登录系统，并且有相应的权限控制。

* 传输安全设计

传输过程中，敏感数据是密文传输的。

# 界面设计原则

## 简易性

界面的简洁是要让用户便于使用、便于理解、并能减少用户发生错误选择的可能性。

## 用户的语言

界面中要使用能反应用户本身的语言，而不是设计者的语言。要用友好性、人性化的提示，言语要友好，减少用户的挫折感，语言是主动式而非被动式，富于提示和启发。

## 记忆负担最小化

人脑不是电脑，在设计界面时必须要考虑人类大脑处理信息的限度。人类的短期记忆也是有限的。所以对用户来说，浏览信息要比记忆信息更容易。这也是用户为何愿意使用带有用户界面的应用而不是只用命令行的原因。

## 一致性

一致性是每一个优秀界面都具备的特点。界面的结构必须清晰且所用的术语要保持一致，风格必须与内容相一致，界面的色调字体也要保持一致。

# 运行设计

## 系统初始化

系统初始化是，具体包括：配置文件中默认配置信息、还有基础数据导入。

## 运行控制

软件运行时有较友好的界面，能够实现用户的数据处理要求。

## 运行结束

用户退出系统时，系统会自动注销该用户的登录会话，释放相关所占用的内存资源，以及释放数据库连接对象资源，将其返还到数据库连接池中等待其他程序调用。

# 系统性能

## 并发性的考虑

关于并发用户数（建议为在线数的10%到20%）。

|  |  |  |
| --- | --- | --- |
| 1 | 系统7\*24小时不间断运行。 | 网络等因素除外 |
| 2 | 一般查询5秒内完成。 |  |
| 3 | 排除网络原因，平均页面打开速度为5秒内完成。 |  |
| 4 | 系统并发量200。 |  |
| 5 | 系统业务平均响应速度在5秒内。 |  |
| 6 | 查询、统计、分析数据准确性100% |  |
| 7 | 数据查询准确率100% |  |
| 8 | 数据吞吐量与硬件配置和网络配置，以及其他系统占用相关。 |  |
| 10 | 系统兼容主流的全版本的中英文输入法。 |  |

# 系统出错处理设计

## 出错信息

在执行程序发生异常时，用户需要进行不同的处理方式。例如：直接将异常抛出；捕获异常，然后执行特定捕获异常机制捕获异常，获取异常类型，然后进行单独处理；对于特定类型的异常跳转到不同的页面，进行显示异常信息等。

在程序中如果发生了异常，如果异常直接抛出，当前正在执行的程序将会停止执行，如果是程序将会跳转到默认的异常处理页面。

## 补救措施

### 数据库备份

通过进行数据库备份。备份软件能够通过网络，对指定的服务器进行数据库备份。在发生系统故障或者人为误操作导致数据丢失或损毁时，备份软件能够及时的恢复数据库，将用户的损失降至最低。

## 系统维护设计

### 环境变量维护

系统对数据库连接信息、日志保存路径、附件保存路径等环境变量以配置文件方式保存。系统管理员能够方便的对系统环境变量进行维护。

# 编码规范

## 命名规范

本系统采用的是springboot+mybatis框架搭建，并在配置文件中做好了详细的配置，所以对包名、类名等的命名规范做了明确的限定。

### 日志记录

日志主要用于配置和记录该构件包中的日志情况。通过日志记录，开发人员可以记录一些关键业务、该构件包中一些错误或者异常信息的情况，便于将来应用运行期间问题的定位、关键业务的审计等工作。

### Java包名的命名规范

实体层包名的命名规范是：com.gzepro.ms.model

持久层包名的命名规范是：com.gzepro.ms.mapper

业务层包名的命名规范是：com.gzepro.ms.service

视图层包名的命名规范是：com.gzepro.ms.controller

### JSP/HTML命名规范

* 页面逻辑(JSP)名称=操作对象名称+操作方法名，多个单词之间用下划线"\_"分割，例如employee\_input.jsp,employee\_list.jsp等,如果名称太长，建议使用词汇的缩写。
* JSP文件名全部要求是小写。
* 含frame页面

<frameset>中<frame>的name属性命名的格式是①xxx.\_②xxx\_③xxx

①xxx部分用来标识当前页面隶属于整个系统中的哪一功能模块。

如：属于ebwebmail则被表示为ebwebmail，其它情况依次类推。

②xxx部分标识当前页面所要完成的功能。

如：完成用户登录的功能则被标识为login，其它情况依次类推。

③xxx部分用来用来表示页面在浏览器窗口所处的位置。

处于浏览器窗口的顶部则标识为top，其它情况依次类推。

例如：ebwebmail\_inbox\_top.jsp

<frame> src属性相应的文件名根据情况建议在原命名规范上用下划线加上所处窗口的位置。

* javascript脚本方法

脚本函数都以①xxx\_②xxx的方式命名。

①xxx对应页面隶属的模块。

②xxx表示函数所要实现的功能(动宾结构)，多个单词用下划线连接。

例如：ebwebmail\_send\_mail()

模块通用的脚本函数必须集合于一个js文件中，在页面上通过<script language="javascript" src="url"></script>形式导入。js文件名命名使用模块名，例如：ebwebmail.js。

如果项目已经提供了公共js脚本，则优先使用公共js脚本中提供的函数。

所有定义方法的<script>元素定义在<head></head>中或</body>后。

* javascript脚本内部变量与参数

单词之间用下划线分隔且全部小写，例如：

var file\_size。

* <form>表单name属性

统一以"form\_"开头，其后加该表单所需收集的信息的作用或动作，例如：form\_file\_upload 或 form\_send\_mail。

* window.open中name参数的命名

javascript的window.open方法中有一个name的参数，浏览器约定同样的名字的窗口只能打开一个，如果程序间名字重复将相互冲突。如果不限制打开窗口数，可以指定''或""（不是null），否则需要加上模块名，例如ebwebmail\_viewmail。

## Java程序代码开规则

以下列出的编码规范，许多都会对代码的可读性和可维护性带来至关重要的作用。99.9％的时间里，程序是用来给人看的，所以，让你的代码有良好的风格和可读性，是非常重要的事情。

### 通用规则

|  |  |
| --- | --- |
| **使用对象** | **规则** |
| 存取方法（getter，setter） | 考虑使用被动初始化的方式。  使用存取方法得到，修改所有的属性。  使用存取方法操作常量。  对于一个集合，添加像集合中插入，删除的方法。 |
| 属 性 | 属性总是要被声明为private。  你可以直接的从类的内部来存取属性，但是一般使用存取方法。  不要使用final static属性，用存取方法来实现。  初始化静态变量。 |
| 类 | 使所有public和protected的接口最少。  按照下面的数许声明属性和方法：   * private attributes * constructors * finalize() * public methods * protected methods * private methods |
| Local variables | 每一行只声明一个变量。  在同一行内用注释记下本地变量。  紧跟”{”之后声明本地变量，也就是说，在代码一开始的时候，就声明。  例外情况:   * Loop counters. e.g.,   for(int i = 0; i < customers.length; i++)   * Exception handling (try-catch). e.g.,   try{  ...  }catch (MalformedURLException e){  ...  }  不要试图用本地变量实现多个目标。 |
| 方 法 | 为代码编写文档。  为代码分段。  .  方法应该不超过一页，太长的方法应该考虑分解。原则：应该在30秒内让人明白你的方法。  尽一切可能限制方法的可访问性。 |

### Import规则

精确的声明每一个import。尽可能的避免使用\*。特别使在调用项目中特定的包的时候。对于比较常用的jdk，或者一些常用的java import可以使用\*。比如 import java.util.\*。

如果要import很多包，那么会进行分类。

Import的顺序：

1. JDK 类和包。.
2. 项目中常用的类或包。
3. 项目中的其他专用包。

## Java编程风格

程序员使用一种良好的编程风格来增强程序的可读性。程序员始终要考虑将来会有别人来对程序进行维护和升级，这部分描述了一些编程风格方面的规范。编程风格可以由程序员自由选择，在本项目中，使用一种统一的编程风格。

### 缩进与空格

使用额外的空格和缩进格式提供非常标准的代码样式。

每一行前以3－5个空格缩进。Use three to five spaces for each indentation level.

在if，while,for,do\_while这些状态时，括号的前后使用空格。

可以任意的使用空格来提高可读性。

属性的声明要排列整齐。

使用空行来帮助区分代码段。

代码示例：

boolean someMethod(boolean existsFlag, String name){

int value = 1; // 使用四个空格

boolean retVal = false; // 把声明对齐

if ( existsFlag == true ) { // 大括号前后使用空格

value = 2; // 又有四个空格

}

}

protected int \_numValues; // 把实例变量对其

protected long \_currCount;

// 区别静态变量

private static final boolean \_DEBUGGING = true;

private static final String \_APP\_NAME = “HelloWorld”;

### 大括号和嵌套风格

一个项目中可能涉及多种的编码风格。不仅对于java编码，对于大括号的排列，也要特别注意多种风格的问题。

如果不考虑风格的话，就用大括号来区别段落(例如遇到if/else, while, for, do-while, switch, try/catch/finally, static)。在一个代码段中，代码应该被处理为标准的缩进距离(3到5个空格)。结尾的大括号应该和开始的大括号在同一级的缩进位置上。

* 对于所有的类，声明从最左边一行开始，一个类的左，右大括号出现在左边的第一行。（也有一种风格，是开始的大括号在类的第一行的最后出现，结束的大括号在最左边的第一行出现。具体的使用，可以根据编译环境的情况选择。按照sun公司的做法，推荐使用第一种方式）。
* 一般的，成对的开始和结束应该在同一列。这样就会使代码段更容易区分。大括号之后，所有的声明和代码都要被缩进至少一级。这里说的一级，通常定义为一个tab键的缩进距离。开发者可以针对自己的开发环境定义一个tab键代表多少个空格。

Example:

public void BalanceDetailServlet (int arg1, int arg2){

int arg3;

*<statements>*

}

* 程序结构必须使用下面的形式：

For loops:

for (<control variable initialization>; < test condition >; <increment control variable>){

<statements>

}

While loops:

while ( <condition> ){

<statements>

}

If constructs:

if ( <condition> ){

< statements >

}

If...else constructs:

if ( <condition> ){

< statements >

}

else{

< statements >

}

If...else if constructs:

if ( <condition> ){

< statements >

}

else if ( <condition> ){

< statements >

}

else{

< statements >

}

Try...catch constructs:

try{

*< statements >*

}catch (Exception\_Type *variable\_name*){

*< statements >*

}

**一直使用大括号**

下面使一个在”if”条件时不使用大括号的例子。下面的代码时合法的，而且经常被用在”if”的情况下。但是会导致错误：

Here’s an example of what can happen if braces are not used on “if” conditions. The following style (using no braces) is legal, and often used on an “if” statement. But it can lead to errors:

//Although this is still legal in Java, it is a very bad example

// that shows how a maintenance programmer might be led astray

if ( someTest() )

\_count = \_count + 1;

如果在维护或者改写代码时，加上了这么一行，那就会产生错误：

In a maintenance or enhancement situation, especially under hurried conditions, a line added using the same style can give unexpected results:

if ( isSomethingOrAnother() )

\_count = \_count + 1; // this line is part of the default block

\_total = \_total + 1; // 这一行也被执行了!!!

所以，哪怕代码段中只有一行，也要使用大括号。

The proper way to prevent this trivial, but potentially very expensive problem from occurring is to *always use braces, even if you have only one line in the block*.

*if ( isSomethingOrAnother() ) { // even if you only have 1 statement!*

*\_count = \_count + 1;*

*}*

### 类的声明

* 总结以上的内容，下面的代码就是一个类声明的例子：

class Circle {

private int myArea;

private int myRadius;

public Circle() {

this(1);

}

public Circle(int newRadius) {

myRadius = newRadius;

calcArea();

}

public int getArea() {

return myArea;

}

public int getRadius() {

return myRadius;

}

public void setRadius(int newRadius) {

myRadius = newRadius;

calcArea();

}

private void calcArea() {

area = pow((PI \* myRadius),2);

}

}

### 例外处理

* 一定要对例外做出处理。至少要把错误信息打印出来。
* 例外的处理，可以根据框架的特性，在类的不同继承层次进行处理。每一层要尽可能多的捕捉异常，但是，每一层次都应该把他捕捉不到的错误推向高层进行处理。
* 不要用例外处理机制做其他的事情。例如，进行程序流转的控制等等。

### 语句的处理

* 较长的语句（>80字符）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

msg = "aaaaaaaaaaaaaaaa"

+"bbbbbbbbbbbbbbbb"

+"cccccccccccccccc";

* 循环、判断等语句中若有较长的表达式或语句，则要进行适应的用()划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，操作数应该放在 == 逻辑运算符的左边，以防止漏写 = 。

if ( (sampDate < currentDate)

&& (0 == this.getCountNum())

&& (0 == counter ) ) {

... // program code

}

* 不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

ect.length = 0; rect.width = 0;

应如下书写

rect.length = 0;

rect.width = 0;

## 注释

### 注释规则

为了确保自己的代码可以被别人看懂，请务必在代码中添加注释。

包括：

1 对自己定义的类和方法一定要进行注释。

2 对关键的sql语句进行注释。

3 对每次的关键修改进行注释。

4 对自己不明确的地方进行注释。

5 自己新建的类要用中文署名。

使用编译环境提供的工具来添加类的注释和对文件进行注释。

# 微服务

## 什么是微服务

微服务得从两个方面去理解，什么是"微"、什么是"服务"， 微 狭义来讲就是体积小、著名的"2 pizza 团队"很好的诠释了这一解释（2 pizza 团队最早是亚马逊 CEO Bezos提出来的，意思是说单个服务的设计，所有参与人从设计、开发、测试、运维所有人加起来 只需要2个披萨就够了 ）。 而所谓服务，一定要区别于系统，服务一个或者一组相对较小且独立的功能单元，是用户可以感知最小功能集。

## 微服务设计原则

**单一职责原则**

意思是每个微服务只需要实现自己的业务逻辑就可以了，比如订单管理模块，它只需要处理订单的业务逻辑就可以了，其它的不必考虑。

**服务自治原则**

* 意思是每个微服务从开发、测试、运维等都是独立的，包括存储的数据库也都是独立的，自己就有一套完整的流程，我们完全可以把它当成一个项目来对待。不必依赖于其它模块。

**轻量级通信原则**

* 首先是通信的语言非常的轻量，第二，该通信方式需要是跨语言、跨平台的，之所以要跨平台、跨语言就是为了让每个微服务都有足够的独立性，可以不受技术的钳制。

**接口明确原则**

* 由于微服务之间可能存在着调用关系，为了尽量避免以后由于某个微服务的接口变化而导致其它微服务都做调整，在设计之初就要考虑到所有情况，让接口尽量做的更通用，更灵活，从而尽量避免其它模块也做调整。

## 微服务优势与缺点

### 11.3.1 微服务优势

**易于开发和维护**

* 由于微服务单个模块就相当于一个项目，开发这个模块我们就只需关心这个模块的逻辑即可，代码量和逻辑复杂度都会降低，从而易于开发和维护。

**启动较快**

* 这是相对单个微服务来讲的，相比于启动单体架构的整个项目，启动某个模块的服务速度明显是要快很多的。

**局部修改容易部署**

* 在开发中发现了一个问题，如果是单体架构的话，我们就需要重新发布并启动整个项目，非常耗时间，但是微服务则不同，哪个模块出现了bug我们只需要解决那个模块的bug就可以了，解决完bug之后，我们只需要重启这个模块的服务即可，部署相对简单，不必重启整个项目从而大大节约时间。

**技术栈不受限**

* 比如订单微服务和电影微服务原来都是用java写的，现在我们想把电影微服务改成nodeJs技术，这是完全可以的，而且由于所关注的只是电影的逻辑而已，因此技术更换的成本也就会少很多。

**按需伸缩**

* 我们上面说了单体架构在想扩展某个模块的性能时不得不考虑到其它模块的性能会不会受影响，对于我们微服务来讲，完全不是问题，电影模块通过什么方式来提升性能不必考虑其它模块的情况。

### 11.3.2 微服务缺点

**运维要求较高**

对于单体架构来讲，我们只需要维护好这一个项目就可以了，但是对于微服务架构来讲，由于项目是由多个微服务构成的，每个模块出现问题都会造成整个项目运行出现异常，想要知道是哪个模块造成的问题往往是不容易的，因为我们无法一步一步通过debug的方式来跟踪，这就对运维人员提出了很高的要求。

**分布式的复杂性**

对于单体架构来讲，我们可以不使用分布式，但是对于微服务架构来说，分布式几乎是必会用的技术，由于分布式本身的复杂性，导致微服务架构也变得复杂起来。

**接口调整成本高**

比如，用户微服务是要被订单微服务和电影微服务所调用的，一旦用户微服务的接口发生大的变动，那么所有依赖它的微服务都要做相应的调整，由于微服务可能非常多，那么调整接口所造成的成本将会明显提高。

**重复劳动**

对于单体架构来讲，如果某段业务被多个模块所共同使用，我们便可以抽象成一个工具类，被所有模块直接调用，但是微服务却无法这样做，因为这个微服务的工具类是不能被其它微服务所直接调用的，从而我们便不得不在每个微服务上都建这么一个工具类，从而导致代码的重复。